A coalgebraic introduction to CSP

Uwe Wolter [1]

FB Informatik
Technische Universität Berlin
D-10587 Berlin, Germany

Abstract

The paper presents a first step of a coalgebraic analysis of the concept of communicating sequential processes as introduced by HOARE in [3]. We make apparent the strong relationship between CSP and partial automata, i.e., special coalgebras. Thereby it turns out that [3] is only dealt with very special automata, namely, with final automata, i.e., automata where the difference between the concepts of state and of process, respectively, disappears. The coalgebraic approach will allow us to develop a proper model theory for process calculi. As first steps in this direction we outline firstly how operations on processes can be generalized in a compatible way to constructions on automata. Secondly, we present a new method for solving recursive process equations. Finally, we discuss that the nondeterminism in CSP can not be modeled based on nondeterministic transition systems usually considered in the coalgebraic literature [5].

1 Introduction

For people usually working on model theory or semantics of formal specifications it becomes often very hard to approach the area of process calculi and process algebras.

There are processes without any physical basis. There is no difference between concepts as machine, agent, process, and state. There is syntax without semantics. There is no difference between processes and process expressions. And so on.

The paper is devoted to make some steps to overcome these difficulties. We show the strong relationship between the concept of *communicating sequential process* [3] and the concept of *partial automaton*. The key concept that allows to bring both concepts into a common perspective is the concept of *coalgebra* [5]. It turns out that CSP is sticked to very special coalgebras, namely to final

[1] Many thanks to HORST REICHEL how started already in 1987 to explain us the ideas of TATSUYA HAGINO.

partial automata without output. The coalgebraic viewpoint, however, offers the opportunity to develop in the future a proper model theory for process calculi.

The paper is written for two kinds of readers. Reader familiar with coalgebraic reasoning as presented, e.g., in [5], can read the paper as an introduction to basic concepts and ideas of CSP. Technically, there will be nothing really new concerning the theory of coalgebras. Reader familiar with CSP or other process calculi should be also able to read the paper. To convince this kind of reader of the practical relevance of abstract category theory we analyse the category theoretic fixed-point construction of final partial automata in some detail.

The author is fully convinced that the theory of coalgebras will be an important link in achieving unifications of theories in computing science as advocated in [4].

The paper is organized as follows. In section 2 we introduce the concept of *deterministic process* according to [3] and try to make apparent the strong relationship to the concept of *deterministic partial automaton*. Thereby, it turns out that processes are related to the *curried* version of partial automata, as studied in [8], thus a coalgebraic treatment of processes appears to be quite natural.

Section 3 explores the insight in [8] that (partial) automata in its curried version should be considered as special coalgebras. We show how the general category theoretic fixed-point construction of final coalgebras applies to deterministic partial automata and that these general construction provides a reasonable model of deterministic processes which turns out to be isomorphic to the Hoare-model.

Section 4 makes evident that interaction of processes is described in [3] in a coalgebraic manner. Moreover, we show that interaction of processes can be generalized to a synchronization of partial automata, and that synchronization of partial automata is compatible with interaction of processes. We close the section with a definition of tests for states in partial automata.

In section 5 we sketch how the crucial insight in [3] that deterministic processes can be completely described by means of traces appears within the coalgebraic approach.

There is no time and space in this paper to give a full analysis of the treatment of nondeterministic processes in [3]. Therefore we will only present in the last section some basic observations and ideas concerning nondeterministic processes and automata to give pointers for further research.

2 Deterministic processes and automata

Fortunately and in contrast to other presentations of processes [3] owns a mathematically rigour which allows to start immediately a more semantically oriented analysis of the proposed concept of process. Firstly, HOARE assumes

for any process P a fixed set A of events in which the process may engage. A is called the *alphabet* of P and is also denoted by αP. The process with alphabet A which never actually engages in any of the events of A is called $STOP_A$.

Secondly, HOARE provides a clean notation for processes. The process which first engages in the event $a \in A = \alpha P$ and then behaves exactly as the process P is denoted by

$$(a \to P) \quad where \quad \alpha(a \to P) = \alpha P \qquad \textbf{(Prefixing)}.$$

Omitting brackets is allowed by the convention that $\to$ is right associative. In such a way a simple vending machine VMA that succesfully serves two customers with chocolate before breaking can be described by the following *process expression*

$$(coin \to choc \to coin \to choc \to STOP_{\alpha VMA})$$

where $\alpha VMA = \{coin, choc\}$. The process which initially engages in either of the *distinct* events $a_1, \ldots, a_n \in A$ and then, after one of these alternative first events a_i has occured, behaves exactly as the process P_i is denoted by

$$(a_1 \to P_1 \mid \cdots \mid a_n \to P_n) \qquad \textbf{(Choice)}$$

where we assume $\alpha P_1 = \ldots = \alpha P_n = A$ and define A to be also the alphabet of $(a_1 \to P_1 \mid \cdots \mid a_n \to P_n)$. Note, that the process denoted by the process expression $(a_1 \to P_1 \mid \cdots \mid a_n \to P_n)$ is deterministic as long as the processes $P_1, \ldots, P_n$ are deterministic since the events $a_1, \ldots, a_n$ are required to be distinct.

A machine VMB that serves either chocolate or toffee before breaking can be described now by the process expression

$$(coin \to (choc \to STOP_{\alpha VMB} \mid tof \to STOP_{\alpha VMB}))$$

where $\alpha VMB = \{coin, choc, tof\}$.

Thirdly, HOARE states that every deterministic process P with alphabet A may be regarded as a *function* F with a domain $B \subseteq A$, defining the set of events in which the process P is initially prepared to engage; and for each a in B, the deterministic process $F(a)$ defines the future behaviour of the process P if the first event was a. This means that every deterministic process $P \in DP_A$ can be uniquely described by a partial function $F : A \nrightarrow DP_A$ with domain $\mathsf{dom}(F) = B$ where DP_A stands for the set of all deterministic processes with alphabet A.

Globally considered, HOARE assumes, in such a way, the existence of a bijective mapping

$$next_A : DP_A \to [A \nrightarrow DP_A]$$

where $[A \nrightarrow DP_A]$ denotes the set of all partial functions from A into DP_A. $STOP_A$, e.g., is the process uniquely determined by the condition $\mathsf{dom}(next_A(STOP_A)) = \emptyset$. RUN_A, i.e., the deterministic process which at all times can engage in any event of A, can be described uniquely by the conditions $\mathsf{dom}(next_A(RUN_A)) = A$ and $next_A(RUN_A)(a) = RUN_A$ for all $a \in A$.

Taking into account the idea of automaton we see immediately that the set of all deterministic processes with alphabet A can be seen as the set of states of an infinite deterministic partial automaton without output. Traditionally [1], a *deterministic partial automaton without output* is defined to be a triple $\mathcal{M} = (I, S, d)$ with I a set of *input symbols*, S a set of *states* and $d : S \times I \nrightarrow S$ a partial *state transition function*. It is well-known, however, that for any such partial function there is an equivalent *curried* version, i.e., a total function $\lambda(d) : S \rightarrow [I \nrightarrow S]$ with $i \in \mathsf{dom}(\lambda(d)(s))$ iff $(s, i) \in \mathsf{dom}(d)$ for all $s \in S$, $i \in I$, and $\lambda(d)(s)(i) = d(s, i)$ for all $i \in \mathsf{dom}(\lambda(d)(s))$. In such a way an automaton $\mathcal{M}$ can be described equivalently using the curried version of d by the tripel $(I, S, \lambda(d))$ as pointed out in [8].

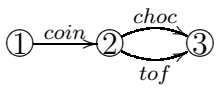
That HOARE's concept of deterministic process can be really reflected by a partial automaton $(A, DP_A, next_A)$ will be justified now by considering the *mathematical model* of deterministic processes in [3]: A deterministic process with alphabet A is defined to be any *prefix closed* subset P of A^*, i.e., any (non-empty) subset $P \in A^*$ which satisfies the two conditions $\langle\rangle \in P$, and $(\forall s, t \in A^* : s\hat{}t \in P \Rightarrow s \in P)$, where $\langle\rangle \in P$ denotes the empty *trace* (finite sequence) and $s\hat{}t$ the *catenation* of traces. The process $STOP_A$ is modeled in this way by the set $\{\langle\rangle\}$ and RUN_A is given by A^* itself. The domain of $next_A(P)$ is denoted in [3] by P^0 and defined by $\mathsf{dom}(next_A(P)) = \{a \mid \langle a\rangle \in P\}$. $next_A(P)(a)$ for any $a \in P^0 = \mathsf{dom}(next_A(P))$ is denoted in [3] by $P(a)$ and defined by $next_A(P)(a) = \{t \mid \langle a\rangle\hat{}t \in P\}$.

¿From now on let DP_A be the set of all prefix closed subsets of A^* and the partial automaton $\mathcal{HM}_A = (A, DP_A, next_A)$ will be called the *Hoare-model* of deterministic processes with alphabet A. Note, that $next_A$ is bijective indeed since we can assign to any partial function $F : A \nrightarrow DP_A$ the prefix closed set $next_A^{-1}(F) = \{\langle\rangle\} \cup \{\langle a\rangle\hat{}t \mid a \in \mathsf{dom}(F) \wedge t \in F(a)\}$.

After realizing that partial automata can serve as a starting point for developing a proper model theory of process calculi it may be interesting to reflect on the meaning of process expressions as, e.g., $VMB = (coin \rightarrow (choc \rightarrow STOP_A \mid tof \rightarrow STOP_A))$ with $A = \{coin, choc, tof\}$.

Firstly, as suggested in [3], it can be interpreted as a "userfriendly" syntactic notation of the prefix closed set $P_{VMB} = \{\langle\rangle, \langle coin\rangle, \langle coin, choc\rangle, \langle coin, tof\rangle\}$ of traces, i.e., as denoting the element P_{VMB} of DP_A (compare also next section).

Secondly, however, we can take VMB as a syntactic presentation of a finite partial automaton $\mathcal{M}_{VMB} = (A, S, t_{VMB})$ with $S = \{1, 2, 3\}$ and $t_{VMB} : S \rightarrow [A \nrightarrow S]$ given by $\mathsf{dom}(t_{VMB}(1)) = \{coin\}$, $\mathsf{dom}(t_{VMB}(2)) = \{choc, tof\}$, $\mathsf{dom}(t_{VMB}(3)) = \emptyset$, and $t_{VMB}(1)\,(coin) = 2$, $t_{VMB}(2)(choc) = t_{VMB}(2)(tof) = 3$. This partial automaton can be depicted as follows (compare section 1.2 in [3] on pictorial representation of processes)



To make the translation of a process expression E into a partial automaton $\mathcal{M}_E$ unambiguous we could use the subexpressions of E to denote the states of $\mathcal{M}_E$ as, e.g., $(coin \rightarrow (choc \rightarrow STOP_A \mid tof \rightarrow STOP_A))$ instead of 1, $(choc \rightarrow STOP_A \mid tof \rightarrow STOP_A)$ instead of 2, and $STOP_A$ instead of 3. Note, that this idea was the reason to identify the codomains of the two arrows starting from 2. Note, further, that this idea would bring us more close to the *labelled transition systems* used in [6] to reason about processes.

In the next section we will see that the Hoare-model $\mathcal{HM}_A$ of deterministic processes can be characterized by being a final object in the category of all deterministic partial automata with alphabet (set of input symbols) A. This means, that there exists for any deterministic partial automaton $\mathcal{M} = (A, S, t : S \rightarrow [A \nrightarrow S])$ a unique *automata morphism* $\tau_{\mathcal{M}} : \mathcal{M} \rightarrow \mathcal{HM}_A$, i.e., a total mapping $\tau_{\mathcal{M}} : S \rightarrow DP_A$ such that $\tau_{\mathcal{M}}(t(s)(a)) = next_A(\tau_{\mathcal{M}}(s))(a)$ for all $s \in S$ and $a \in A$, where the left hand side of the equation is defined if, and only if, the right hand side is defined. Note, that this condition is equivalent to the traditional condition for the uncurried version of automata morphisms.

$$\begin{array}{ccc} S & \xrightarrow{\quad t \quad} & [A \nrightarrow S] \\ {\scriptstyle \tau_{\mathcal{M}}}\downarrow & & \downarrow{\scriptstyle -;\tau_{\mathcal{M}}} \\ DP_A & \xrightarrow{\quad next_A \quad} & [A \nrightarrow DP_A] \end{array} \qquad \begin{array}{ccc} S \times A & \xrightarrow{\quad \lambda^{-1}(t) \quad} & S \\ {\scriptstyle \tau_{\mathcal{M}} \times id_A}\downarrow & & \downarrow{\scriptstyle \tau_{\mathcal{M}}} \\ DP_A \times A & \xrightarrow{\quad \lambda^{-1}(next_A) \quad} & DP_A \end{array}$$

In our example $\tau_{\mathcal{M}_{VMB}} : \{1, 2, 3\} \rightarrow DP_A$ will map 1 to P_{VMB}, 2 to $\{\langle\rangle, \langle choc\rangle, \langle tof\rangle\}$, and 3 to $\{\langle\rangle\}$.

Note, that both interpretations of a process expression are compatible because the translation of a process expression E into a deterministic partial automaton $\mathcal{M}_E$ points out implicitly an initial state in $\mathcal{M}_E$, namely, the state that corresponds to the whole expression E, and this state will be maped by $\tau_{\mathcal{M}_E}$ to the process P_E obtained by the "process interpretation" of the expression. For our example we have, e.g., $\tau_{\mathcal{M}_{VMB}}(1) = P_{VMB}$.

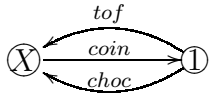
Using prefixing and choice we can only build process expressions that denote finite deterministic processes. To be able to describe syntactically infinite processes HOARE introduces recursion. Let X be a *process variable* and $F(X)$ be a process expression build on X by prefixing and choice using events from a fixed set A. The idea in [3] is that $F(X)$ defines a map $[\![F]\!] : DP_A \rightarrow DP_A$ such that the *recursive process equation* $X = F(X)$ can be taken as the syntactic description of a deterministic process if there is exactly one *fixed-point* of $[\![F]\!]$. HOARE proves that this is the case as long as $F(X)$ is *guarded*, i.e., as long as there is at least one occurrence of $\rightarrow$ in $F(X)$. The unique fixed-point is refered to as $\mu X : A.F(X)$.

A machine VMC with alphabet $A = \{coin, choc, tof\}$, e.g., that either serves chocolate or toffee in a loop can be described by the recursive equation

$$X = (coin \rightarrow (choc \rightarrow X \mid tof \rightarrow X))$$

where the corresponding unique fixed-point is given by all traces from A^*

with *coin* at each odd position and either *choc* or *tof* at each even position. The translation of process expressions into deterministic partial automata as sketched above offers, however, a new method to assign uniquely a process to a recursive equation $X = F(X)$: We can construct a finite partial automaton $\mathcal{M}_{X=F(X)} = (A, S, t)$ with $X \in S$ such that the image of X w.r.t. the unique automata morphism $\tau_{\mathcal{M}_{X=F(X)}} : \mathcal{M}_{X=F(X)} \to \mathcal{HM}_A$ can be taken as the deterministic process described by the equation $X = F(X)$. For our example VMC the corresponding partial automaton $\mathcal{M} = (A, S, t)$ with $S = \{X, 1\}$ can be depicted as follows



That for guarded expressions $\tau_{\mathcal{M}_{X=F(X)}}(X)$ equals the unique fixed-point of $[\![F]\!]$ according to [3] will hopefully become clear in the next section. Note, however, that we will assign, in contrast to [3], also a unique process to the equation $X = X$, namely, $STOP_A$, i.e., the "least fixed-point" of $Id : DP_A \to DP_A$. Finally, we want to mention that the process $RUN_A = A^*$ can be represented by a "one-state" partial automaton (A, S, t) with $S = \{X\}$, $\mathsf{dom}(t(X)) = A$, and $t(X) : A \to S$ the unique total function from A into $\{X\}$.

3 Final coalgebras

For a functor $T : SET \to SET$ a *T-coalgebra* is a pair (S, t) consisting of a set S, the *carrier* of the coalgebra, and a mapping $t : S \to T(S)$. A *T-homomorphism* $f : (S_1, t_1) \to (S_2, t_2)$ between two T-coalgebras (S_1, t_1) and (S_2, t_2) consists of a mapping $f : S_1 \to S_2$ which commutes with the operations: $t_1; T(f) = f; t_2$.

$$\begin{array}{ccc} S_1 & \xrightarrow{t_1} & T(S_1) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle T(f)} \\ S_2 & \xrightarrow{t_2} & T(S_2) \end{array}$$

To apply this definition to deterministic partial automata we have only to check that the assignment $S \mapsto [A \nrightarrow S]$ extends to a functor $A_{\rightarrow} : SET \to SET$. For this we assign to any mapping $f : S_1 \to S_2$ the mapping $A_{\rightarrow}(f) : [A \nrightarrow S_1] \longrightarrow [A \nrightarrow S_2]$ with

$$A_{\rightarrow}(f)(g) =_{def} g; f \quad \text{for all functions } g \in [A \nrightarrow S_1].$$

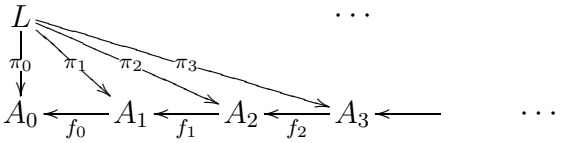
It is easy to check that this defines really a functor $A_{\rightarrow} : SET \to SET$. Now the concepts "deterministic partial automata with alphabet A" and "$A_{\rightarrow}$-coalgebra" turn out to be obviously equivalent.

Because the functor $A_{\rightarrow} : SET \to SET$ is ω^{op}-continuous [7], i.e., preserves limits of ω^{op}-chains we can fortunately, use the category theoretic version of the *least fixed-point construction* [9] to construct the final $A_{\rightarrow}$-

coalgebra: The limit $(L, (\pi_i : L \to A_i)_{i\in\mathbb{N}})$ of an ω^{op}-chain

$$A_0 \xleftarrow{f_0} A_1 \xleftarrow{f_1} A_2 \xleftarrow{f_2} A_3 \longleftarrow \quad \cdots$$

in SET can be described canonically by all infinite sequences $\langle a_0, a_1, a_2, \ldots\rangle$ such that $a_i \in A_i$ and $f_i(a_{i+1}) = a_i$ for all $i \in \mathbb{N}$. The mapping $\pi_i : L \to A_i$ projects $\langle a_1, a_1, a_2, \ldots\rangle$ to the i-th component a_i thus we have $\pi_i = \pi_{i+1}; f_i$, for all $i \in \mathbb{N}$.

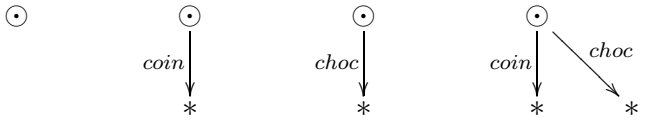


The carrier NF_A of the intended final $A_{\to}$-coalgebra will be given now by the limit $(NF_A, (\pi_i : NF_A \to A^i_{\to}(1))_{i\in\mathbb{N}})$ of the following ω^{op}-chain

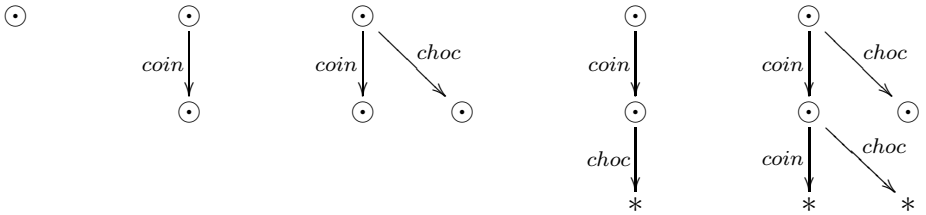
$$1 \xleftarrow{!} A_{\to}(1) \xleftarrow{A_{\to}(!)} A^2_{\to}(1) \xleftarrow{A^2_{\to}(!)} A^3_{\to}(1) \longleftarrow \quad \cdots$$

obtained by applying successively the functor $A_{\to}$ to the unique mapping from $A_{\to}(1)$ into the singleton set $1 = \{*\}$, i.e., into the final object of the category SET.

To see that NF_A is something, we are already familiar with, we firstly consider the elements of $A^i_{\to}(1)$, which can be refered to as *nested functions* of depth less than or equal to i. For $A = \{coin, choc\}$, e.g., $A_{\to}(1) = [A \nrightarrow 1]$ has four functions as elements and by representing a function by its *graph* we get $A_{\to}(1) = \{\emptyset, \{(coin, *)\}, \{(choc, *)\}, \{(coin, *), (choc, *)\}\}$. A pictorial representation could look as follows



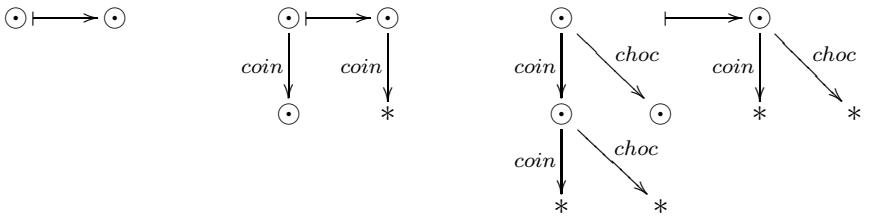
$A^2_{\to}(1) = [A \nrightarrow [A \nrightarrow 1]]$ has $(4+1)^2 = 25$ elements as , e.g., $\emptyset$, $\{(coin, \emptyset)\}$, $\{(coin, \emptyset), (choc, \emptyset)\}$, $\{(coin, \{(choc, *)\})\}$, and $\{(coin, \{(coin, *), (choc, *)\}), (choc, \emptyset)\}$ wich can be depicted by



Now it becomes apparent that process expressions build by prefixing and choice can be taken as an appropriate notation for graphs of nested functions of finite depth. Thereby, $STOP_A$ stands for the graph $\emptyset$ of the fully undefined

function and variables X represent the graph of an "unknown" function $*$. In such a way the elements $g_1 = \{(coin, \emptyset)\}$ and $g_2 = \{(coin, \{(coin, *), (choc, *)\}), (choc, \emptyset)\}$ of $A^2_{\rightarrow}(1)$, e.g., correspond to the process expressions $(coin \rightarrow STOP_A)$ and $(coin \rightarrow (coin \rightarrow X_1 \mid choc \rightarrow X_2) \mid choc \rightarrow STOP_A)$, respectively. Taking into account the pictorial representation we could also consider the elements of $A^i_{\rightarrow}(1)$ as *synchronization trees* with depth less than or equal to i [6,10].

$! : A_{\rightarrow}(1) \rightarrow 1$ maps each function in $A_{\rightarrow}(1) = [A \nrightarrow 1]$ to $*$ thus $A_{\rightarrow}(!) = \_;! : [A \nrightarrow [A \nrightarrow 1]] \rightarrow [A \nrightarrow 1]$ maps each $g \in [A \nrightarrow [A \nrightarrow 1]]$ to $g;! \in [A \nrightarrow 1]$ with $\mathsf{dom}(g;!) = \mathsf{dom}(g)$ and $(g;!)(a) = !(g(a)) = *$ for all $a \in \mathsf{dom}(g;!)$. In general $A^i_{\rightarrow}(!) : A^{i+1}_{\rightarrow}(1) \rightarrow A^i_{\rightarrow}(1)$ just cuts the (possibly empty) $(i+1)$-th layer of a tree with depth less than or equal to $(i+1)$ where the information that a cutting has taken place at depth i is announced by writing $*$ at the corresponding node. For our example we have $A_{\rightarrow}(!)(\emptyset) = \emptyset$, $A_{\rightarrow}(!)(g_1) = \{(coin, *)\}$, and $A_{\rightarrow}(!)(g_2) = \{(coin, *), (choc, *)\}$, i.e., we have the following transformation of trees

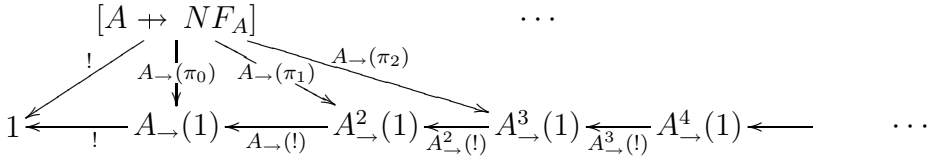


The elements of NF_A are by construction infinite sequences $\langle *, g_1, g_2, g_3, \ldots \rangle$ of nested functions (synchronization trees) such that $A^i_{\rightarrow}(!)(g_{i+1}) = g_i$, i.e., g_i equals g_j up to depth i if $i < j$. In such a way $\langle *, g_1, g_2, g_3, \ldots \rangle$ appears to be an *approximation* of a possibly infinite process (compare law L3 on page 96 in [3]). $\langle *, g_1, g_2, g_3, \ldots \rangle$ represents a finite process if $g_i = g_{i+1}$ for some $i \in \mathbb{N}$ and thus $g_i = g_j$ for all $i \leq j$. Note, that in case $g_i = g_{i+1}$ there is no $*$ in the graph of g_i, i.e., no variable in the corresponding process expression.

The ω^{op}-continuity of $A_{\rightarrow} : SET \rightarrow SET$ entails that $(A_{\rightarrow}(NF_A), (A_{\rightarrow}(\pi_i) : A_{\rightarrow}(NF_A) \rightarrow A^{i+1}_{\rightarrow}(1))_{i \in \mathbb{N}})$ is a limit of the ω^{op}-chain

$$A_{\rightarrow}(1) \xleftarrow{A_{\rightarrow}(!)} A^2_{\rightarrow}(1) \xleftarrow{A^2_{\rightarrow}(!)} A^3_{\rightarrow}(1) \xleftarrow{A^3_{\rightarrow}(!)} A^4_{\rightarrow}(1) \longleftarrow \quad \cdots$$

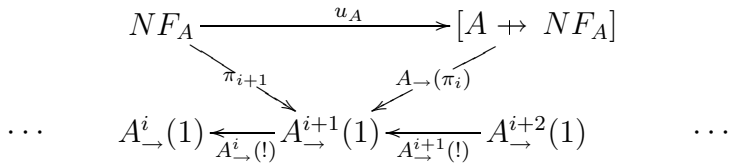
Since there is only one mapping from $A_{\rightarrow}(NF_A) = [A \nrightarrow NF_A]$ into 1 we have trivially $A_{\rightarrow}(\pi_0);! = !$ thus we obtain a further limit diagram for the original ω^{op}-chain



The limit properties of both diagrams ensure the existence of a unique mapping $u_A : NF_A \to [A \nrightarrow NF_A]$ with

$$u_A \,;\, A_{\to}(\pi_i) = \pi_{i+1} \quad \text{for all } i \in \mathbb{N}, \tag{1}$$

and, moreover, it is ensured that this mapping is bijective, i.e., an isomorphism in SET.



The intended *coalgebraic model* of deterministic processes is now provided by the $A_{\to}$-coalgebra $\mathcal{CM}_A = (NF_A, u_A)$

Remark 3.1 Note, that the category theoretic fixed-point construction provides a kind of "external" approximation of processes. That is, a process P is identified with the sequence $\langle *, g_1, g_2, g_3, \ldots \rangle$ of its finite approximations, where the g_i's are not processes but real approximations of processes. Please bear in mind that an open branch in g_i is indicated by $*$ and not by $\emptyset$.

There is no need for an "internal" approximation of processes as used, e.g., in [3], i.e., for defining a partial order on processes thus the finite processes can serve as finite approximations of infinite processes.

To see that the coalgebraic model and the Hoare-model are isomorphic we have to analyse how the mapping $u_A : NF_A \to [A \nrightarrow NF_A]$ works. Let be given a sequence $P = \langle *, g_1, g_2, g_3, \ldots \rangle$ in NF_A. The image of P w.r.t. u_A has to be a partial function $u_A(P) : A \nrightarrow NF_A$ thus we have firstly to determine the domain of $u_A(P)$. For this we have to bear in mind that all partial functions $g_{i+1} \in A^{i+1}_{\to}(1) = [A \nrightarrow A^i_{\to}(1)]$, $i \in \mathbb{N}$ have the same domain because

$$g_{i+1} = A^{i+1}_{\to}(!)(g_{i+2}) = A_{\to}(A^i_{\to}(!))(g_{i+2}) = g_{i+2} \,;\, A^i_{\to}(!)$$

with $A^i_{\to}(!)$ a total mapping for all $i \in \mathbb{N}$. That the domain of u_A equals this common domain of the components g_{i+1} of P is forced by equation 1, which implies for all $i \in \mathbb{N}$

$$g_{i+1} = \pi_{i+1}(P) = A_{\to}(\pi_i)(u_A(P)) = u_A(P); \pi_i \tag{2}$$

and thus $\mathsf{dom}(u_A(P)) = \mathsf{dom}(u_A(P); \pi_i) = \mathsf{dom}(g_{i+1})$ since π_i is total.

Next, we have to define for any $a \in \mathsf{dom}(u_A(P))$ a sequence $u_A(P)(a) = \langle *, g^a_1, g^a_2, g^a_3, \ldots \rangle \in NF_A$. Equation 2, however, tells that $g^a_i = \pi_i(u_A(P)(a)) = g_{i+1}(a)$ thus we are done.

Theorem 3.2 *The Hoare-model $\mathcal{HM}_A = (DP_A, next_A)$ and the coalgebraic model $\mathcal{CM}_A = (NF_A, u_A)$ are isomorphic $A_{\to}$-coalgebras, i.e., there exists a bijective mapping $appr_A : DP_A \to NF_A$ such that the following diagram*

commutes

$$\begin{array}{ccc} DP_A & \xrightarrow{next_A} & [A \nrightarrow DP_A] \\ {\scriptstyle appr_A}\downarrow & & \downarrow{\scriptstyle -;appr_A} \\ NF_A & \xrightarrow{u_A} & [A \nrightarrow NF_A] \end{array}$$

Proof sketch: According to [3] any prefix closed set of traces $P \in DP_A$ can be described uniquely by the infinite sequence $\langle P \restriction 0, P \restriction 1, P \restriction 2, \ldots\rangle$ of prefix closed sets of *bounded traces* where $P \restriction n =_{def} \{s \in P \mid |s| \leq n\}$, i.e., $P \restriction n$ contains all traces from P with length at most n. Prefix closedness ensures that these sequences $\langle P \restriction 0, P \restriction 1, P \restriction 2, \ldots\rangle$ are in one-to-one correspondence to sequences of nested functions $\langle *, g_1, g_2, g_3, \ldots\rangle \in NF_A$. That this translation of P into an approximating sequence $\langle *, g_1, g_2, g_3, \ldots\rangle$ of nested functions is compatible with $next_A$ and u_A can be checked straightforwardly using the results above and the results stated in [3]. □

Remark 3.3 Note, that exactly the prefix closedness ensures that P can be described by an approximating sequence $\langle P \restriction 0, P \restriction 1, P \restriction 2, \ldots\rangle$ of finite processes, and that this approximation is the basis in [3] to prove uniqueness of the solution of a guarded recursive process equation.

The coalgebraic model $\mathcal{CM}_A = (NF_A, u_A)$ is final in the category of $A_{\rightarrow}$-coalgebras by construction [9]. Since $\mathcal{HM}_A$ is isomorphic to $\mathcal{CM}_A$ we have

Corollary 3.4 *$\mathcal{CM}_A = (NF_A, u_A)$ and $\mathcal{HM}_A = (DP_A, next_A)$ are final $A_{\rightarrow}$-coalgebras.*

To justify the claim in section 2 that our new method of solving recursive equations $X = F(X)$, which is based on the finality of $\mathcal{HM}_A$ or $\mathcal{CM}_A$, respectively, is strongly related to the fixed-point construction in [3], we have to look more close to the proof of the finality of $\mathcal{CM}_A$.

Let $\mathcal{M} = (S, t : S \rightarrow [A \nrightarrow S])$ be an arbitrary $A_{\rightarrow}$-coalgebra. What we are interested in is to determine the process that starts in a state $s \in S$. That is, we have to analyse step-by-step which states can be reached from s by which transitions.

The unfolding of the state transition function $t : S \rightarrow [A \nrightarrow S]$ gives the following sequence of commutative diagrams

$$\begin{array}{ccccccccc} S & \xrightarrow{t} & A_{\rightarrow}(S) & \xrightarrow{A_{\rightarrow}(t)} & A^2_{\rightarrow}(S) & \xrightarrow{A^2_{\rightarrow}(t)} & A^3_{\rightarrow}(S) & \longrightarrow & \cdots \\ {\scriptstyle !_S}\downarrow & & {\scriptstyle A_{\rightarrow}(!_S)}\downarrow & & {\scriptstyle A^2_{\rightarrow}(!_S)}\downarrow & & {\scriptstyle A^3_{\rightarrow}(!_S)}\downarrow & & \\ 1 & \xleftarrow[!]{} & A_{\rightarrow}(1) & \xleftarrow[A_{\rightarrow}(!)]{} & A^2_{\rightarrow}(1) & \xleftarrow[A^2_{\rightarrow}(!)]{} & A^3_{\rightarrow}(1) & \longleftarrow & \cdots \end{array}$$

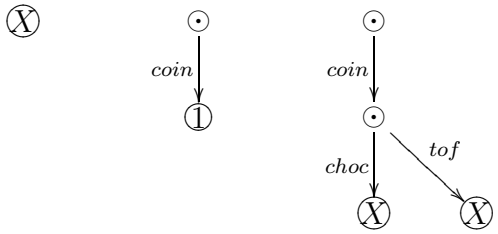
where the left-most rectangle is commutative since there is only one mapping from S into 1 and all other rectangles are stepwise images of the first one.

$t : S \to A_\to(S)$ tells for any state $s \in S$ which states can be reached from s in one step by which transition. $A^i_\to(t) : A^i_\to(S) \to A^{i+1}_\to(S)$ describes how arbitrary sequences of transitions of length i, i.e., sequences not taking into account the restrictions made by t, can be continued according to t in the next step. Starting in a state $s \in S$ we obtain in such a way an infinite sequence

$$unfold_{\mathcal{M}}(s) =_{def} \langle s, t^s_1, t^s_2, \ldots\rangle \quad \text{with } t^s_{i+1} =_{def} A^i_\to(t)(t^s_i) \in A^{i+1}_\to(S)$$

for all $i \in \mathbb{N}$, i.e., with $t^s_1 = t(s) \in A_\to(S)$, $t^s_2 = A_\to(t)(t(s)) \in A^2_\to(S)$, ..., where t^s_i represents all sequences of transitions in $\mathcal{M}$ of length atmost i starting in s. Moreover, t^s_i tells which states are reached by sequences of length exactly i. The states visited in between are forgotten in t^s_i.

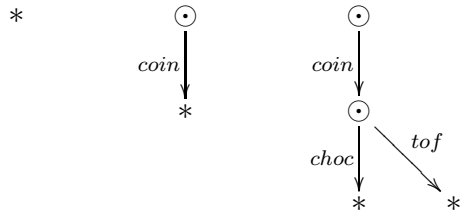
For the vending machine VMC and the (initial) state X we could depict, e.g., the first three elements of $unfold_{VMC}(X)$ as follows



Finally, we consider the abstraction of $unfold_{\mathcal{M}}(s)$ into a process. The mapping $!_S : S \to 1$ identifies all states to $*$ thus $A^i_\to(!_S) : A^i_\to(S) \to A^i_\to(1)$ just forgets the information, which states are reached by sequences of length i and keeps only the infomation that sequences of length i may be continued. We obtain now for any $s \in S$ an infinite sequence

$$proc_{\mathcal{M}}(s) =_{def} \langle *, g^s_1, g^s_2, \ldots\rangle \quad \text{with} \quad g^s_i =_{def} A^i_\to(!_S)(t^s_i) \text{ for all } i \geq 1\,.$$

The first three elements of $proc_{VMC}(X)$, e.g., are



The commutativity of the above diagrams and the definition of t^s_{i+1} and g^s_{i+1}, respectively, entail for all $i \in \mathbb{N}$

$$\begin{aligned} g^s_i &= A^i_\to(!_S)(t^s_i) = A^i_\to(!)(A^{i+1}_\to(!_S)(A^i_\to(t)(t^s_i))) = A^i_\to(!)(A^{i+1}_\to(!_S)(t^s_{i+1})) \\ &= A^i_\to(!)(g^s_{i+1}) \end{aligned}$$

thus $proc_{\mathcal{M}}(s)$ becomes indeed a process, i.e., an element of NF_A. This means that we have constructed by $proc_{\mathcal{M}}(s)$ the process starting in state $s \in S$. Globally this provides a mapping $proc_{\mathcal{M}} : S \to NF_A$. That this mapping constitutes a $A_\to$-homomorphism $proc_{\mathcal{M}} : \mathcal{M} \to \mathcal{CM}_A$ and that this $A_\to$-

homomorphism is unique can be proved straightfowardly according to the limit construction of NF_A and the ω^{op}-continuity of the functor $A_{\rightarrow} : SET \rightarrow SET$.

4 Interaction and concurrency

Firstly, HOARE discusses the interaction of processes P and Q with the same alphabet $\alpha P = \alpha Q$. He defines a process $P \parallel Q$ with $\alpha(P \parallel Q) = \alpha P = \alpha Q$ which behaves like the system composed of P and Q interacting in *lock-step synchronization*, i.e., any occurence of events requires simultaneous participation of both the processes involved. To model this kind of interaction we have to define a mapping $\_ \parallel \_ : NF_A \times NF_A \rightarrow NF_A$.

In the last section we have seen that $\mathcal{CM}_A = (NF_A, u_A)$ is the final $A_{\rightarrow}$-coalgebra, i.e., the final partial automaton with alphabet A. This offers a canonical way to define mappings from an arbitrary set S into NF_A [5]. We have only to construct a $A_{\rightarrow}$-coalgebra $\mathcal{M} = (S, t)$ with carrier S. Then, by finality of $\mathcal{CM}_A$, there exists a unique $A_{\rightarrow}$-homomorphism $proc_{\mathcal{M}} : \mathcal{M} \rightarrow \mathcal{CM}_A$. The only problem will be to design $\mathcal{M}$ in such a way that the underlying mapping $proc_{\mathcal{M}} : S \rightarrow NF_A$ becomes the intended one.

To define interaction of processes we construct a $A_{\rightarrow}$-coalgebra

$$\mathcal{SYN}_A = (NF_A \times NF_A, syn_A : NF_A \times NF_A \longrightarrow [A \nrightarrow NF_A \times NF_A])$$

which can be considered as the automaton obtained by synchronizing the partial automaton $\mathcal{CM}_A$ with itself: For any pair of processes $(P, Q) \in NF_A \times NF_A$ we define

$$\mathsf{dom}(syn_A(P, Q)) =_{def} \mathsf{dom}(u_A(P)) \cap \mathsf{dom}(u_A(Q))$$

and for all $a \in \mathsf{dom}(syn_A(P, Q))$ we set

$$syn_A(P, Q)(a) =_{def} (u_A(P)(a), u_A(Q)(a)) .$$

The final $A_{\rightarrow}$-homomorphism $proc_{\mathcal{SYN}_A} : \mathcal{SYN}_A \rightarrow \mathcal{CM}_A$ due to section 3 makes the following diagram commutative

$$\begin{array}{ccc} NF_A \times NF_A & \xrightarrow{syn_A} & [A \nrightarrow NF_A \times NF_A] \\ {\scriptstyle proc_{\mathcal{SYN}_A}}\downarrow & & \downarrow{\scriptstyle \_ ; proc_{\mathcal{SYN}_A}} \\ NF_A & \xrightarrow{u_A} & [A \nrightarrow NF_A] \end{array}$$

That is, for each pair $(P, Q) \in NF_A \times NF_A$ the equation

$$u_A(proc_{\mathcal{SYN}_A}(P, Q)) = syn_A(P, Q) ; proc_{\mathcal{SYN}_A}$$

is required. For any event $z \in \mathsf{dom}(syn_A(P, Q))$ this means that

$$u_A(proc_{\mathcal{SYN}_A}(P, Q))(z) = proc_{\mathcal{SYN}_A}(u_A(P)(z), u_A(Q)(z)) .$$

Using the notation in [3] the last condition turns into the equation $(P \parallel Q)(z) = P(z) \parallel Q(z)$ thus it becomes apparent that the coalgebraic definition of $proc_{\mathcal{SYN}_A} : NF_A \times NF_A \rightarrow NF_A$ is equivalent to the requirements stated in law 4, page 67 in [3] for the interaction operator $\_ \parallel \_ : NF_A \times NF_A \rightarrow NF_A$.

Since $proc_{\mathcal{SYN}_A}$ is uniquely defined by the above conditions we can be sure that $proc_{\mathcal{SYN}_A}$ is indeed the intended interaction operator $\_ \parallel \_$. Using coinductive proof techniques [5] we could prove now the other laws for interaction stated in [3].

Secondly, HOARE describes the concurrent interaction of processes P and Q with different alphabets $\alpha P \neq \alpha Q$. Only events that are in both their alphabets, i.e., in the intersection $\alpha P \cap \alpha Q$, are required to synchronize. However, events in the alphabet of P but not in the alphabet of Q may occur independently of Q whenever P engages in them. Similarly, Q may engage alone in events which are in the alphabet of Q but not of P. In such a way the alphabet of the process $P \parallel Q$ will be the union $\alpha P \cup \alpha Q$ of the alphabets of the component processes. Note, that the use of overstrokes in [6] is another technique to fix which events in different sets A and B have to synchronize.

Let be given now two alphabets A and B. The coalgebraic definition of the intended mapping $\_ \parallel \_ : NF_A \times NF_B \to NF_{A \cup B}$ can be extracted from law 7, page 71 in [3]. The synchronization of $\mathcal{CM}_A$ and $\mathcal{CM}_B$ provides a partial automaton

$\mathcal{SYN}_{A,B} = (NF_A \times NF_B, syn_{A,B} : NF_A \times NF_B \longrightarrow [A \cup B \nrightarrow NF_A \times NF_B])$

with alphabet $A \cup B$ as follows: For any pair of processes $(P, Q) \in NF_A \times NF_B$ we define

$$\begin{aligned} &\mathsf{dom}(syn_{A,B}(P,Q)) \\ &=_{def} \mathsf{dom}(u_A(P)) \setminus B \cup \mathsf{dom}(u_A(P)) \cap \mathsf{dom}(u_B(Q)) \cup \mathsf{dom}(u_B(Q)) \setminus A \end{aligned}$$

and for any $c \in \mathsf{dom}(syn_{A,B}(P,Q))$ we set

$$syn_{A,B}(P,Q)(c) =_{def} \begin{cases} (u_A(P)(c), Q) & , c \in \mathsf{dom}(u_A(P)) \setminus B \\ (u_A(P)(c), u_B(Q)(c)) & , c \in \mathsf{dom}(u_A(P)) \cap \mathsf{dom}(u_B(Q)) \\ (P, u_A(Q)(c)) & , c \in \mathsf{dom}(u_B(Q)) \setminus A \end{cases}$$

The final $(A \cup B)_{\to}$-homomorphism $proc_{\mathcal{SYN}_{A,B}} : \mathcal{SYN}_{A,B} \to \mathcal{CM}_{A \cup B}$ provides the intended concurrent interaction operator $\_ \parallel \_ : NF_A \times NF_B \to NF_{A \cup B}$. Note, that obviously $\mathcal{SYN}_{A,A} = \mathcal{SYN}_A$.

The coalgebraic definition of the concurrent interaction operator suggests a straightforward generalization of synchronization to arbitrary partial automata.

Definition 4.1 For any partial automata $\mathcal{M}_1 = (S_1, t_1 : S_1 \to [A \nrightarrow S_1])$ and $\mathcal{M}_2 = (S_2, t_1 : S_2 \to [B \nrightarrow S_2])$ we define the corresponding *synchronized automaton*

$$\mathcal{SYN}_{\mathcal{M}_1,\mathcal{M}_2} = (\, S_1 \times S_2 \,,\, syn_{\mathcal{M}_1,\mathcal{M}_2} : S_1 \times S_2 \longrightarrow [A \cup B \nrightarrow S_1 \times S_2] \,)$$

as follows: For any $(s_1, s_2) \in S_1 \times S_2$ we define

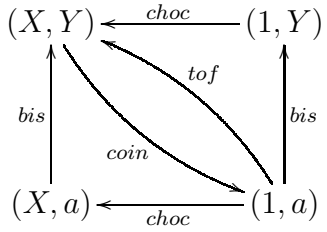
$$\begin{aligned} &\mathsf{dom}(syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1,s_2)) \\ &=_{def} \mathsf{dom}(t_1(s_1)) \setminus B \cup \mathsf{dom}(t_1(s_1)) \cap \mathsf{dom}(t_2(s_2)) \cup \mathsf{dom}(t_2(s_2)) \setminus A \end{aligned}$$

and for any $c \in \mathsf{dom}(syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1,s_2))$ we set

$$syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1,s_2)(c) =_{def} \begin{cases} (t_1(s_1)(c), s_2) & , c \in \mathsf{dom}(t_1(s_1)) \setminus B \\ (t_1(s_1)(c), t_2(s_2)(c)), & c \in \mathsf{dom}(t_1(s_1)) \cap \mathsf{dom}(t_2(s_2)) \\ (s_1, t_2(s_2)(c)) & , c \in \mathsf{dom}(t_2(s_2)) \setminus A \end{cases}$$

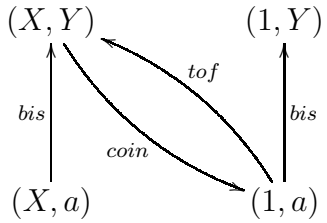
As an example we synchronize the vending machine VMC from section 2 with alphabet $A = \{coin, choc, tof\}$ and a customer with alphabet $B = \{coin, tof, bis\}$ described by the recursive equation $Y = (coin \to (tof \to Y \mid bis \to Y))$. After paying a coin the customer decides between having a toffee or a biscuit instead. The corresponding partial automata can be depicted by

and the synchronization of both automata gives



That is, after the customer was able to pay a coin he may decide for toffee and the machine can deliver a toffee at the same time. If he decides for biscuit the machine will serve up later on a chocolate. Or, even worth, the machine may decide to give him a chocolate and he has to interpret this as his own decision for biscuit to have a second chance to get a toffee.

Note, that by simply changing the alphabet of the customer to $B = \{coin, tof, bis, choc\}$ we would obtain a synchronized automaton with a deadlock



Now it turns out that the concurrent interaction of processes exactly describes how the processes in a synchronized automaton $\mathcal{SYN}_{\mathcal{M}_1,\mathcal{M}_2}$ can be obtained by composing the processes of the single automata $\mathcal{M}_1$ and $\mathcal{M}_2$. That is, synchronization of automata is compatible with interaction of processes as stated in

Theorem 4.2 *For any partial automata* $\mathcal{M}_1 = (S_1, t_1 : S_1 \to [A \nrightarrow S_1])$,

$\mathcal{M}_2 = (S_2, t_1 : S_2 \to [B \nrightarrow S_2])$, *and any pair of states* $(s_1, s_2) \in S_1 \times S_2$ *we have that*

$$proc_{\mathcal{SYN}_{\mathcal{M}_1,\mathcal{M}_2}}(s_1, s_2) = proc_{\mathcal{M}_1}(s_1) \parallel proc_{\mathcal{M}_2}(s_2)$$

Proof: Since $\_ \parallel \_ : NF_A \times NF_B \to NF_{A \cup B}$ is given by the final $(A \cup B)_{\to}$-homomorphism $proc_{\mathcal{SYN}_{A,B}} : \mathcal{SYN}_{A,B} \to \mathcal{CM}_{A \cup B}$ it suffices to show that the mapping $proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2} : S_1 \times S_2 \to NF_A \times NF_B$ constitutes a $(A \cup B)_{\to}$-homomorphism $proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2} : \mathcal{SYN}_{\mathcal{M}_1,\mathcal{M}_2} \to \mathcal{SYN}_{A,B}$.

$$\begin{array}{ccc} S_1 \times S_2 & \xrightarrow{syn_{\mathcal{M}_1,\mathcal{M}_2}} & [A \cup B \nrightarrow S_1 \times S_2] \\ \downarrow{\scriptstyle proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2}} & & \downarrow{\scriptstyle \_ ; proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2}} \\ NF_A \times NF_B & \xrightarrow{syn_{A,B}} & [A \cup B \nrightarrow NF_A \times NF_B] \end{array}$$

The required equality $proc_{\mathcal{SYN}_{\mathcal{M}_1,\mathcal{M}_2}} = proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2} ; proc_{\mathcal{SYN}_{A,B}}$ is then ensured by the uniqueness of final homomorphisms.

We have to show that for any pair $(s_1, s_2) \in S_1 \times S_2$ the equality

$$syn_{A,B}(proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2}(s_1, s_2)) = syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1, s_2) ; proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2} \quad (3)$$

holds. Since $proc_{\mathcal{M}_1} : \mathcal{M}_1 \to \mathcal{CM}_A$ is a $A_{\to}$-homomorphism we have for $s_1 \in S_1$ the equality

$$u_A(proc_{\mathcal{M}_1}(s_1)) = t_1(s_1) ; proc_{\mathcal{M}_1} \quad (4)$$

and since $proc_{\mathcal{M}_2} : \mathcal{M}_2 \to \mathcal{CM}_B$ is a $B_{\to}$-homomorphism we have for $s_2 \in S_2$

$$u_B(proc_{\mathcal{M}_2}(s_2)) = t_2(s_2) ; proc_{\mathcal{M}_2} \; . \quad (5)$$

According to the equations 4 and 5, the totality of the mappings $proc_{\mathcal{M}_1}$, $proc_{\mathcal{M}_2}$, and the definition of $\mathcal{SYN}_{\mathcal{M}_1,\mathcal{M}_2}$ and $\mathcal{SYN}_{A,B}$, respectively, we can firstly show that the domain of both functions in equation 3 are equal:

$$\begin{aligned} &\mathsf{dom}(syn_{A,B}(proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2}(s_1, s_2))) \\ &= \mathsf{dom}(syn_{A,B}(proc_{\mathcal{M}_1}(s_1), proc_{\mathcal{M}_2}(s_2))) \\ &= \mathsf{dom}(u_A(proc_{\mathcal{M}_1}(s_1))) \setminus B \cup \\ &\quad \mathsf{dom}(u_A(proc_{\mathcal{M}_1}(s_1))) \cap \mathsf{dom}(u_B(proc_{\mathcal{M}_2}(s_2))) \cup \mathsf{dom}(u_B(proc_{\mathcal{M}_2}(s_2))) \setminus A \\ &= \mathsf{dom}(t_1(s_1) ; proc_{\mathcal{M}_1}) \setminus B \cup \\ &\quad \mathsf{dom}(t_1(s_1) ; proc_{\mathcal{M}_1}) \cap \mathsf{dom}(t_2(s_2) ; proc_{\mathcal{M}_2}) \cup \mathsf{dom}(t_2(s_2) ; proc_{\mathcal{M}_2}) \setminus A \\ &= \mathsf{dom}(t_1(s_1)) \setminus B \cup \mathsf{dom}(t_1(s_1)) \cap \mathsf{dom}(t_2(s_2)) \cup \mathsf{dom}(t_2(s_2)) \setminus A \\ &= \mathsf{dom}(syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1, s_2)) \\ &= \mathsf{dom}(syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1, s_2) ; proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2}) \end{aligned}$$

Secondly, we show the equality 3 for all $c \in \mathsf{dom}(t_1(s_1)) \setminus B = \mathsf{dom}(u_A(proc_{\mathcal{M}_1}(s_1))) \setminus B$. According to definition of $syn_{A,B}$, the equality 4, and the definition of $syn_{\mathcal{M}_1,\mathcal{M}_2}$ we obtain

$$\begin{aligned} &syn_{A,B}(proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2}(s_1, s_2))(c) \\ &= syn_{A,B}(proc_{\mathcal{M}_1}(s_1), proc_{\mathcal{M}_2}(s_2))(c) \\ &= (u_A(proc_{\mathcal{M}_1}(s_1))(c), proc_{\mathcal{M}_2}(s_2)) \end{aligned}$$

$$= (proc_{\mathcal{M}_1}(t_1(s_1)(c)), proc_{\mathcal{M}_2}(s_2))$$
$$= proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2}(syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1, s_2)(c))$$
$$= (syn_{\mathcal{M}_1,\mathcal{M}_2}(s_1, s_2) ; proc_{\mathcal{M}_1} \times proc_{\mathcal{M}_2})(c)$$

The other cases can be proved analogously. □

The general synchronization of partial automata can be used, further, to define a *test* for states in partial automata. Namely, the test if a process can successfully run starting in a certain state (compare [2]). We consider a partial automaton $\mathcal{M} = (S, t : S \to [A \nrightarrow S])$. Then a process $P \in NF_A$ can *successfully run* starting in a state $s \in S$ if $proc_{\mathcal{SYN}_{\mathcal{CM}_A,\mathcal{M}}}(P, s) = P$.

5 Traces and runs

Let be given a set A of events. Then the set A^∞ of all finite and infinite traces can be made into a $A_\rightarrow$-coalgebra $\mathcal{TR}_A = (A^\infty, pop_A : A^\infty \to [A \nrightarrow A^\infty])$. We set $pop_A(\langle\rangle) =_{def} \emptyset$ for the empty traces $\langle\rangle \in A^\infty$ and for non-empty traces $tr = \langle a_0, a_1, a_2, \ldots\rangle$ we define $\mathsf{dom}(pop_A(tr)) =_{def} \{a_0\}$ and $pop_A(tr)(a_0) =_{def} \langle a_1, a_2, \ldots\rangle$. The final $A_\rightarrow$-homomorphism $proc_{\mathcal{TR}_A} : \mathcal{TR}_A \to \mathcal{CM}_A$ will transform any trace $\langle a_0, a_1, a_2, \ldots\rangle$ into the sequence $\langle *, \{(a_0, *)\}, \{(a_0, \{(a_1, *)\})\}, \{(a_0, \{(a_1, \{(a_2, *)\})\})\}, \ldots\rangle$ of nested functions that can be seen as the sequence of prefixes of $\langle a_0, a_1, a_2, \ldots\rangle$. Obviously, the mapping $proc_{\mathcal{TR}_A} : A^\infty \to NF_A$ is injective thus $\mathcal{TR}_A$ can be considered as a *subcoalgebra* of $\mathcal{CM}_A$ [7].

Now, we can test for any state in a partial automaton if a trace can run successfully starting in this state. That is, for any partial automaton $\mathcal{M} = (S, t)$ with alphabet A the synchronization of $\mathcal{M}$ with $\mathcal{TR}_A$ will become a linear automaton thus the $A_\rightarrow$-homomorphism $proc_{\mathcal{SYN}_{\mathcal{TR}_A,\mathcal{M}}} : \mathcal{SYN}_{\mathcal{TR}_A,\mathcal{M}} \longrightarrow \mathcal{CM}_A$ can be decomposed into a $A_\rightarrow$-homomorphism $test_{\mathcal{M}} : \mathcal{SYN}_{\mathcal{TR}_A,\mathcal{M}} \longrightarrow \mathcal{TR}_A$ followed by the $A_\rightarrow$-embedding $proc_{\mathcal{TR}_A} : \mathcal{TR}_A \to \mathcal{CM}_A$. For the final partial automaton $\mathcal{CM}_A$ these tests are exhaustive in the following sense.

Proposition 5.1 *Let be given a set A of events. Then for any processes $P, Q \in NF_A$ the following assertions are equivalent:*

(i) $P = Q$

(ii) $test_{\mathcal{CM}_A}(tr, P) = test_{\mathcal{CM}_A}(tr, Q)$ *for all traces* $tr \in A^\infty$.

(iii) *For all traces* $tr \in A^\infty$ *holds that*

$$test_{\mathcal{CM}_A}(tr, P) = tr \quad \textit{iff} \quad test_{\mathcal{CM}_A}(tr, Q) = tr$$

Runs, i.e., sequences of transitions in a partial automaton which can not be continued, can be defined by the same technique. We consider the subcoalgebra $\mathcal{RUN}_A = (A^\omega, pop_A : A^\omega \to [A \nrightarrow A^\omega])$ of $\mathcal{TR}_A$ with A^ω the set of all infinite traces. The $A_\rightarrow$-homomorphism $proc_{\mathcal{SYN}_{\mathcal{RUN}_A,\mathcal{M}}} : \mathcal{SYN}_{\mathcal{RUN}_A,\mathcal{M}} \longrightarrow \mathcal{CM}_A$ can be decomposed into a $A_\rightarrow$-homomorphism $run_{\mathcal{M}} : \mathcal{SYN}_{\mathcal{RUN}_A,\mathcal{M}} \longrightarrow \mathcal{TR}_A$ followed by the $A_\rightarrow$-homomorphism $proc_{\mathcal{TR}_A} : \mathcal{TR}_A \to \mathcal{CM}_A$. Then

for each infinite trace $r \in A^\omega$ the possibly finite trace $run_{\mathcal{M}}(r, s) \in A^\infty$ is a run in $\mathcal{M}$ starting in state $s \in S$.

6 Nondeterministic processes and automata

The crucial observation is that CSP is not based on nondeterministic transition systems usually considered in the coalgebraic literature [5]. That is, nondeterminism in CSP can be modeled neither with unbounded nondeterminism based on the power set construction $\mathcal{P}(A \times S)$ nor with finite nondeterminism based on the finite power set construction $\mathcal{P}_f(A \times S)$. CSP deals instead with a weaker kind of nondeterminism, namely with a kind of *nondeterministic filter*.

If a nondeterministic partial automaton being in a certain state can engage in an event then the state reached in the next step will be uniquely determined by the event. Nondeterminism is restricted to the possibility to decide locally in each state which events will be accepted or, alternatively, refused for the next step. That is, even in case we can engage in an event it may be that we can not carry out this event because it was decided before not to accept this event for the next step.

HOARE uses families of sets of refused events to model this kind of nondeterminism. He claims, however, that the use of families of sets of accepted events would be equivalent thus we decide for this possibility.

The *nondeterministic partial automata* we have to consider in such a way have the following structure

$$\mathcal{M} = (\, S\,,\, t : S \to \mathcal{P}(\mathcal{P}(A)) \times [A \nrightarrow S] \,)$$

where A is the alphabet of $\mathcal{M}$. For any state $s \in S$ we will denote the first component of $t(s)$ by $acc(t(s))$ and the second component will be denoted, in abuse of notation, also by $t(s)$. This means that we are dealt with $A_{\underrightarrow{n}}$-coalgebras for the functor $A_{\underrightarrow{n}} : SET \to SET$ with $A_{\underrightarrow{n}}(S) =_{def} \mathcal{P}(\mathcal{P}(A)) \times [A \nrightarrow S]$ for each set S and $A_{\underrightarrow{n}}(f) =_{def} id_{\mathcal{P}(\mathcal{P}(A))} \times A_{\rightarrow}(f)$ for each mapping f.

This functor is also ω^{op}-continuous thus we can construct, analogously to the case of deterministic automata, for each alphabet A a final $A_{\underrightarrow{n}}$-coalgebra

$$\mathcal{NCM}_A = (\, AT_A\,,\, v_A : AT_A \to \mathcal{P}(\mathcal{P}(A)) \times [A \nrightarrow AT_A] \,).$$

The elements of AT_A are again infinite sequences $\langle *, g_1, g_2, g_3, \ldots \rangle$ where the components g_i can be seen as *acceptance trees* of depth at most i [2], i.e., as trees with a family of sets of events at each node. $CHAOS_A \in AT_A$, e.g., that is the most nondeterministic process which at all times can engage in any event of A and at the same time refuse any event of A, can be described uniquely by the conditions $acc(CHAOS_A) = \mathcal{P}(A)$, $\mathsf{dom}(v_A(CHAOS_A)) = A$ and $v_A(CHAOS_A)(a) = CHAOS_A$ for all $a \in A$.

Note, that we are not modelling exactly the nondeterministic processes in [2] or [3], respectively, since we don't require the corresponding saturation

properties for the acceptance (refusal) sets. One point of further research will be to analyse aim and object of these saturation conditions.

We can assign to any deterministic partial automaton $\mathcal{M} = (S, t : S \to [A \nrightarrow S])$ a corresponding nondeterministic partial automaton

$$\mathcal{M}_n = (S, t_n : S \to \mathcal{P}(\mathcal{P}(A)) \times [A \nrightarrow S]) \text{ with } t_n(s) =_{def} (\{\mathsf{dom}(t(s))\}, t(s))$$

for all $s \in S$.Note, that the corresponding embedding according to [3] would take instead of the singleton family of acceptance sets $\{\mathsf{dom}(t(s))\}$ the family $\mathcal{P}(A \backslash \mathsf{dom}(t(s)))$ of refusal sets. That is the the resulting nondeterministic partial automaton would own a proper internal nondeterminism which, however, can never be observed from outside.

One of the main achievements of the coalgebraic approach is that we can generalize operations on processes to compatible constructions on partial automata. A very good point in doing this is that we are able to justify in such a way our intuition about operations on processes.

As an example we will look therefore how the *nondeterministic or-operator* $P \sqcap Q$ on processes may be generalized to partial automata. The nondeterministic coupling $\mathcal{M}_1 \sqcap \mathcal{M}_2$ of two automata is given by the disjoint union of $\mathcal{M}_1$ and $\mathcal{M}_2$ plus some branching points where a decision for automaton $\mathcal{M}_1$ or $\mathcal{M}_2$ can be made. Thereby the decision between $\mathcal{M}_1$ and $\mathcal{M}_2$ will be postponed if both automata can engage in the same event.

Definition 6.1 For any nondeterministic partial automata $\mathcal{M}_1 = (S_1, t_1 : S_1 \to \mathcal{P}(\mathcal{P}(A)) \times [A \nrightarrow S_1])$ and $\mathcal{M}_2 = (S_2, t_2 : S_2 \to \mathcal{P}(\mathcal{P}(A)) \times [A \nrightarrow S_2])$ with the same alphabet A the nondeterministic partial automaton

$$\mathcal{M}_1 \sqcap \mathcal{M}_2 = (S, t : S \to \mathcal{P}(\mathcal{P}(A)) \times [A \nrightarrow S])$$

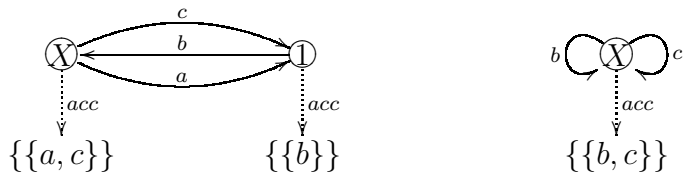
is given by $S =_{def} S_1 + S_1 \times S_2 + S_2$ and for each $s \in S$ we set

- $s \in S_1$: $acc(t(s)) =_{def} acc(t_1(s))$ and $t(s) =_{def} t_1(s) ; in_1$ with injection $in_1 : S_1 \to S_1 + S_1 \times S_2 + S_2$
- $s \in S_2$: $acc(t(s)) =_{def} acc(t_2(s))$ and $t(s) =_{def} t_2(s) ; in_2$ with injection $in_2 : S_2 \to S_1 + S_1 \times S_2 + S_2$
- $s = (s_1, s_2) \in S_1 \times S_2$: $acc(t(s_1, s_2)) =_{def} acc(t_1(s_1)) \cup acc(t_2(s_2))$
  $\mathsf{dom}(t(s_1, s_2)) =_{def} \mathsf{dom}(t_1(s_1)) \cup \mathsf{dom}(t_2(s_2))$
  and for all $a \in \mathsf{dom}(t(s_1, s_2))$

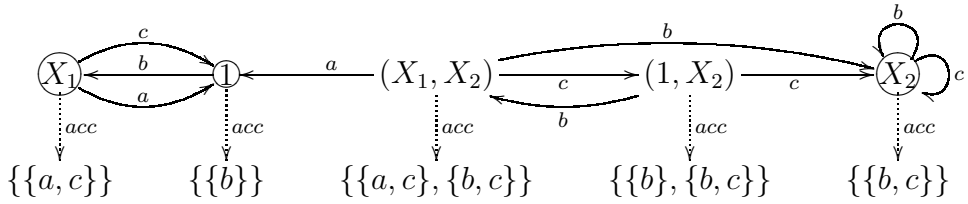
$$t(s_1, s_2)(a) =_{def} \begin{cases} t_1(s_1)(a) & , a \in \mathsf{dom}(t_1(s_1)) \setminus \mathsf{dom}(t_2(s_2)) \\ (t_1(s_1)(a), t_2(s_2)(a)) & , a \in \mathsf{dom}(t_1(s_1)) \cap \mathsf{dom}(t_2(s_2)) \\ t_2(s_2)(a) & , a \in \mathsf{dom}(t_2(s_2)) \setminus \mathsf{dom}(t_1(s_1)) \end{cases}$$

As an example we consider the (deterministic) automaton $\mathcal{M}_1$ with alphabet $A = \{a, b, c\}$ given by the recursive equation $X = (a \to (b \to X) \mid c \to (b \to X))$ and the (deterministic) automaton $\mathcal{M}_2$ with the same alphabet A

given by $X = (b \to X \mid c \to X)$. That is, we consider the two automata



The nondeterministic coupling $\mathcal{M}_1 \sqcap \mathcal{M}_2$ is than given by

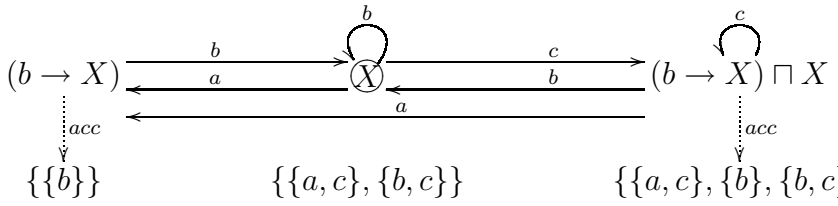


Starting in the branching state (X_1, X_2) we may decide for the set $\{a, c\}$ of events and thus implicitly for $\mathcal{M}_1$. This means, firstly, that we are not allowed to engage in event b and thus to reach the corresponding copy of $\mathcal{M}_2$. If we engage in event a our decision for $\mathcal{M}_1$ will be registered thus we will live from now on in the corresponding copy of $\mathcal{M}_1$. If we engage in event c our decision for $\mathcal{M}_1$ can not be detected thus we get a second chance to choose now in the branching state $(1, X_2)$ again between $\mathcal{M}_1$ and $\mathcal{M}_2$.

Besides interpreting $\_ \sqcap \_$ as an operator on processes and as a constructor on automata we could also use $\_ \sqcap \_$ to build a new kind of process expressions and thus a new kind of recursive equations. Using the expressions describing $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively, we could build, e.g., the following recursive equation

$$X = (a \to (b \to X) \mid c \to (b \to X)) \sqcap (b \to X \mid c \to X).$$

which would represent the following nondeterministic partial automaton $\mathcal{M}$



Note, that the difference between $\mathcal{M}_1 \sqcap \mathcal{M}_2$ and $\mathcal{M}$ is related to the observation in [3] that recursion does not distribute over $\_ \sqcap \_$.

It seems, unfortunately, that our coalgebraic approach will not meet exactly the approach in [3]. Our approach ensures that the solutions of recursive equations are compatible with the embedding of deterministic processes/automata into nondeterministic processes/automata. In [3] the fixed-point construction for deterministic equations starts with $STOP_A$ and the fixed-point construction for nondeterministic processes starts with $CHAOS_A$. The embedding of $STOP_A$, however, does not provide $CHAOS_A$ thus we will

get for a deterministic recursive equation two different solutions within the domain of nondeterministic processes. This discrepancy will be another point of further research.

What is really missing in the present paper is the analysis of *bisimulations* in CSP. We hope that there will be time and space for a corresponding sequel of our coalgebraic introduction to CSP.

References

[1] H. Ehrig, K.D. Kiermeier, H.J. Kreowski, and W. Kühnel. *Universal Theory of Automata*. Teubner, Stuttgart, 1974.

[2] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[3] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[4] C.A.R. Hoare. Unification of Theories: AChallenge for Computing Science. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specification*, pages 49–57. 11th Workshop on Specification of Abstract Data Types, WADT11, Oslo Norway, September 1995, Springer, LNCS 1130, 1996.

[5] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of EATCS*, 62:222–259, 1997.

[6] R. Milner, editor. *Communication and Concurrency*. Prentice Hall International, 1989.

[7] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. Technical Report CS-R9652, CWI, 1996.

[8] J.J.M.M. Rutten. Automata and coinduction (an exersice in coalgebra). Technical Report SEN-R9803, CWI, 1998.

[9] M.B. Smyth and G.D. Plotkin. The category theoretic solution of recursive domain equations. *SIAM Journ. Comput.*, 11:761–783, 1982.

[10] G. Winskel and M. Nielsen. Models for concurrency. Technical Report DAIMI PB 463, Aarhus University, 1993. to appear as a chapter in the Handbook of Logic in Computer Science.